Lecture 17: NP-Complete Problems and the Cook-Levin Theorem
Time-Bounded Complexity Classes

Turing machine M has time complexity $O(t(n))$ if there is a $c > 0$ such that for all inputs $x$, M running on $x$ halts within $c \cdot t(|x|) + c$ steps

Definition:
$\text{TIME}(t(n)) = \{ L' | \text{there is a Turing machine } M \text{ with time complexity } O(t(n)) \text{ so that } L' = L(M) \}$

$= \{ L' | L' \text{ is a language decided by a Turing machine with } \leq c \cdot t(n) + c \text{ running time, for some } c \geq 1 \}$
The Time Hierarchy Theorem

Intuition: If you get more time to compute, then you can solve strictly more problems.

Theorem: For all “reasonable” $f, g : \mathbb{N} \rightarrow \mathbb{N}$ where for all $n$, $g(n) > n^2 f(n)^2$, $\text{TIME}(f(n)) \not\subseteq \text{TIME}(g(n))$

Proof Idea: Diagonalization with a clock
Make a TM $N$ that on input $M$, simulates the TM $M$ on input $M$ for $f(|M|)$ steps, then flips the answer.

Show: $L(N)$ cannot have time complexity $f(n)$
And $N$ can be implemented to run in time $g(n)$
$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$

Polynomial Time

The analogue of “decidability” in the world of complexity theory
Definition: $\text{NTIME}(t(n)) =$
\[
\{ L \mid L \text{ is decided by an } O(t(n)) \text{ time } \text{nondeterministic Turing machine} \}\]

Note: $\text{TIME}(t(n)) \subseteq \text{NTIME}(t(n))$

Is $\text{TIME}(t(n)) = \text{NTIME}(t(n))$ for all $t(n)$?

**THIS IS AN OPEN QUESTION!**

What can be done in “short” $\text{NTIME}$ that cannot be done in “short” $\text{TIME}$?
NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)

Nondeterministic Polynomial Time

The analogue of “recognizability” in complexity
P
Computation

accept or reject

n^k

NP
Computation

accept

reject

exp(n^k)
Theorem: $L \in NP \iff$ There is a constant $k$ and polynomial-time TM $V$ such that

$$L = \{ x \mid \exists y \in \Sigma^* \ [ |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts} \} \}$$

Proof: (1) If $L = \{ x \mid \exists y \ |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts} \} \}$ then $L \in NP$

Given the poly-time TM $V$, our NP machine for $L$ is:

$N(x)$: Nondeterministically guess $y$. Run $V(x,y)$

(2) If $L \in NP$ then

$$L = \{ x \mid \exists y \ |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts} \} \}$$

Let $N$ be a nondet. poly-time TM that decides $L$. Define a TM $V(x,y)$ which accepts

$\iff y$ encodes an accepting computation history of $N$ on $x$
A language is in NP if and only if there are “polynomial-length proofs” for membership in the language.

\[ P = \text{the problems that can be efficiently solved} \]

\[ \text{NP} = \text{the problems where proposed solutions can be efficiently verified} \]

Is \( P = \text{NP} \)?

Can problem solving be automated?
P = NP?
So how do we get a handle on a problem that we have no idea how to resolve?

Understand its consequences!  
*Understand its meaning!*

Try to better understand NP problems!

In computability theory, we related problems by mapping reductions....
Polynomial Time Reductions

$f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function
if there is a poly-time Turing machine $M$ that on
every input $w$, halts with just $f(w)$ on its tape

Language $A$ is poly-time reducible to language $B$,
written as $A \leq_p B$,
if there is a poly-time computable $f : \Sigma^* \rightarrow \Sigma^*$ so that:
\[ w \in A \iff f(w) \in B \]

We say: $f$ is a polynomial time reduction from $A$ to $B$

Note: there is a $k$ such that for all $w$, $|f(w)| \leq k|w|^k$
f converts any string \( w \) into a string \( f(w) \) such that \( w \in A \iff f(w) \in B \)
Theorem: If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
Theorem: If \( A \leq_p B \) and \( B \in P \), then \( A \in P \)

Proof: Let \( M_B \) be a poly-time TM that decides \( B \). Let \( f \) be a poly-time reduction from \( A \) to \( B \).

We build a machine \( M_A \) that decides \( A \) as follows:

\[
M_A = \text{On input } w,
\]

1. Compute \( f(w) \)
2. Run \( M_B \) on \( f(w) \), output its answer

\[
w \in A \iff f(w) \in B
\]
Theorem: If $A \leq_p B$ and $B \in \text{NP}$, then $A \in \text{NP}$

Proof: Analogous...
Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

Theorem: If $A \leq_p B$ and $B \in NP$, then $A \in NP$

Corollary: If $A \leq_p B$ and $A \notin P$, then $B \notin P$

Question: What are the “hardest” NP problems under this partial ordering $\leq_p$?

Does there even exist a “hardest” NP problem??
Definition: A language B is NP-complete if:

1. B $\in$ NP
2. Every A in NP is poly-time reducible to B
   That is, $A \leq_p B$
   When this is true, we say “B is NP-hard”

On homework, you showed
A language L is recognizable iff $L \leq_m A_{TM}$

$A_{TM}$ is "complete for recognizable languages":
$A_{TM}$ is recognizable, and for all recognizable L, $L \leq_m A_{TM}$
Suppose $L$ is NP-Complete...

If $L \in P$, then $P = NP$!

If $L \notin P$, then $P \neq NP$!
Suppose L is NP-Complete...

Then assuming the conjecture $P \neq NP$,

$L$ is not decidable in $n^k$ time, for every $k$
There exists an NP-complete problem

\[ \text{NHALT} = \{ (N, x, 1^t) \mid \text{Nondeterministic TM } N \text{ accepts input } x \text{ in } \leq t \text{ steps} \} \]

1. \( \text{NHALT} \in \text{NP} \)

Without 1^t, this is undecidable!

Nondeterministically guess a sequence of \( t \) transitions of \( N \), then check that \( N \) following these \( t \) transitions accepts \( x \). Takes time polynomial in \( t \), \|x\|, and \|N\|.

2. Every \( A \) in NP is poly-time reducible to NHALT

“NHALT is NP-hard”

Every \( A \) in NP has a \( p(n) \)-time NTM \( N \) such that

\[ A = \{ x \mid N(x) \text{ accepts} \} \]

Reduction: Map string \( x \) to the string \( (N, x, 1^{p(|x|)}) \).
There are thousands of *natural* NP-complete problems!

Your favorite topic certainly has an NP-complete problem somewhere in it!

Even the other sciences are not safe: biology, chemistry, physics have NP-complete problems too!
A 3cnf-formula has the form:
\[(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)\]

where \(x_1, x_2, \ldots\) are Boolean variables

A 3cnf-formula is satisfiable if there is a setting to the variables that makes the formula true.

\[3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}\]
The Cook-Levin Theorem:  
3SAT is NP-complete

“Simple Logic can encode any NP problem”

1. 3SAT ∈ NP
A satisfying assignment is a “proof” that a 3cnf formula is satisfiable (already done!)

2. 3SAT is NP-hard
Every language in NP can be polynomial-time reduced to 3SAT (complex logical formula)

Corollary: 3SAT ∈ P if and only if P = NP
Theorem (Cook-Levin): 3SAT is NP-complete

Proof Idea:

(1) $3\text{SAT} \in \text{NP}$ (done)

(2) Every language $A$ in NP is polynomial time reducible to $3\text{SAT}$ (this is the challenge)

We give a poly-time reduction from $A$ to $3\text{SAT}$

The reduction converts a string $w$ into a 3cnf formula $\phi$ such that $w \in A$ iff $\phi \in 3\text{SAT}$

For any $A \in \text{NP}$, let $N$ be a nondeterministic TM deciding $A$ in $n^k$ time

Idea: $\phi$ will “simulate” $N$ on $w$
Let \( L(N) \in \text{NTIME}(n^k) \). A tableau for \( N \) on \( w \) is an \( n^k \times n^k \) matrix whose rows are the configurations of \textit{some} computation history of \( N \) on \( w \)

\[
\begin{array}{cccccccc}
\# & q_0 & w_1 & w_2 & \ldots & w_n & \square & \ldots & \square & \# \\
\# & & & & & & & & & \\
\# & & & & & & & & & \\
\# & & & & & & & & & \\
\end{array}
\]

Each "cell" contains an \( \sigma \in Q \cup \Gamma \cup \{\#\} \)
A tableau is accepting if the last row of the tableau has an accept state

Therefore, N accepts string w if and only if there is an accepting tableau for N on w

Given w, we’ll construct a 3cnf formula $\phi$ with $O(|w|^{2k})$ clauses, describing logical constraints that any accepting tableau for N on w must satisfy

The 3cnf formula $\phi$ will be satisfiable if and only if there is an accepting tableau for N on w
Variables of formula $\phi$ will *encode* a tableau

Let $C = Q \cup \Gamma \cup \{ \# \}$ (*constant-sized set!*)

Each cell of a tableau contains a symbol from $C$

$\text{cell}[i,j] = \text{symbol in the cell at row } i \text{ and column } j$

$= \text{the } j\text{th symbol in the } i\text{th configuration}$

For every $i$ and $j \ (1 \leq i, j \leq n^k)$ and for every $s \in C$
we make a Boolean variable $x_{i,j,s}$ in $\phi$

Total number of variables $= |C|n^{2k}$, which is $O(n^{2k})$

The $x_{i,j,s}$ variables represent the cells of a tableau

We will enforce the condition: for all $i, j, s$,

$$x_{i,j,s} = 1 \iff \text{cell}[i,j] = s$$
Idea: Make $\phi$ so that every \textit{satisfying assignment} to the variables $x_{i,j,s}$ corresponds to an \textit{accepting tableau} for N on w (an assignment to all cell[i,j]’s of the tableau)

The formula $\phi$ will be the AND of four CNF formulas:

$$\phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}}$$

$\phi_{\text{cell}}$: for all $i$, $j$, there is a \textit{unique} $s \in C$ with $x_{i,j,s} = 1$

$\phi_{\text{start}}$: the first row of the table equals the \textit{start} configuration of N on w

$\phi_{\text{accept}}$: the last row of the table has an accept state

$\phi_{\text{move}}$: every row is a configuration that yields the configuration on the next row
\( \phi_{\text{start}} : \text{the first row of the table equals the } \textit{start} \text{ configuration of } N \text{ on } w \)

\[
\phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land \\
x_{1,3,w_1} \land x_{1,4,w_2} \land \cdots \land x_{1,n+2,w_n} \land \\
x_{1,n+3,□} \land \cdots \land x_{1,n^k-1,□} \land x_{1,n^k,#}
\]

\[
\begin{array}{cccccccc}
\# & q_0 & w_1 & w_2 & \ldots & w_n & □ & \ldots & □ & \# \\
\# & & & & & & & & & \\
\# & & & & & & & & & \\
\end{array}
\]

\( O(n^k) \) clauses
φ_{accept} : the last row of the table has an accept state

φ_{accept} = \bigvee_{1 \leq j \leq n^k} x_{n^k,j, q_{accept}}
\( \phi_{\text{accept}} \): the last row of the table has an accept state

\[ \phi_{\text{accept}} = \bigvee_{1 \leq j \leq n^k} x_{n^k, j, q_{\text{accept}}} \]

How can we convert \( \phi_{\text{accept}} \) into a 3-cnf formula?

Write the clause \((a_1 \lor a_2 \lor \ldots \lor a_t)\) as

\[(a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land \ldots \land (\neg z_{t-3} \lor a_{t-1} \lor a_t)\]

where \(z_i\) are new variables.

This produces \(O(t)\) new 3cnf clauses, and the new formula is SAT iff the old one is SAT.

\( O(n^k)\) 3cnf clauses
\[ \phi_{cell} : \text{for all } i, j, \text{ there is a } \text{unique } s \in C \text{ with } x_{i,j,s} = 1 \]

\[ \phi_{cell} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \wedge \left( \bigwedge_{s,t \in C} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \quad s \neq t \right) \right] \]

- for all i,j
- at least one \( x_{i,j,s} \) is set to 1
- at most one \( x_{i,j,s} \) is set to 1

\[ O(n^{2k}) \text{ 3cnf clauses} \]
\( \phi_{\text{move}} \) : every row is a configuration that yields the configuration on the next row

Key Question: If one row yields the next row, how many cells can be different between the two rows?

Answer: AT MOST THREE CELLS!
$\phi_{\text{move}}$: every row is a configuration that yields the configuration on the next row

Key Question: If one row yields the next row, how many cells can be different between the two rows?

Answer: AT MOST THREE CELLS!
\( \phi_{\text{move}} \) : every row is a configuration that yields the configuration on the next row

Idea: check that every 2 \( \times \) 3 “window” of cells is legal: consistent with the transition function of N

\[
\begin{array}{cccccccc}
\# & q_0 & w_1 & w_2 & \ldots & w_n & \square & \ldots & \square & \# \\
\# & & & & & & & & & \\
\# & & & & & & & & & \\
\# & i & j & & & & & & & \\
\# & & & & & & & & & \\
\# & & & & & \text{the (i,j) window} & & & & \\
\# & & & & & & & & & \\
\# & & & & & & & & & \\
\end{array}
\]
If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$
which of the following windows are legal?
Key Lemma:
IF  Every window of the tableau is legal, and
    The 1\textsuperscript{st} row is the start configuration of N on w
THEN  for all \( i = 1,\ldots,n^k - 1 \), the \( i \)th row of the tableau is
    a configuration which yields the \((i+1)\)th row.

Proof Sketch: (Strong) induction on \( i \).
The 1\textsuperscript{st} row is a configuration. If it \textit{didn’t} yield the 2\textsuperscript{nd} row, there’s a 2 x 3 “illegal” window on 1\textsuperscript{st} and 2\textsuperscript{nd} rows
Assume rows 1,\ldots,L are all configurations which yield the next row, and assume every window is legal.
If row L+1 did \textit{not} yield row L+2, then there’s a 2 x 3 window along those two rows which is “illegal”
The \((i, j)\) window of a tableau is the tuple \((a_1, \ldots, a_6) \in \mathbb{C}^6\) such that

\[
\begin{array}{ccc}
\text{row } i & \text{col. } j & \text{col. } j+1 & \text{col. } j+2 \\
\hline
a_1 & a_2 & a_3 \\
a_4 & a_5 & a_6 \\
\end{array}
\]
\( \phi_{\text{move}} : \) every row is a configuration that legally follows from the previous configuration

\[
\phi_{\text{move}} = \bigwedge ( \text{the (i, j) window is legal})
\]

\[
1 \leq i \leq n^k - 1 \\
1 \leq j \leq n^k - 2
\]

\( (\text{the (i, j) window is legal}) = \)

\[
\bigvee (a_1, ..., a_6)
\]

(a_1, ..., a_6) is a legal window

\[
\equiv \bigwedge (a_1, ..., a_6)
\]

(a_1, ..., a_6) is NOT a legal window

\[
\equiv (x_{i,j,a_1} \lor x_{i,j+1,a_2} \lor x_{i,j+2,a_3} \lor x_{i+1,j,a_4} \lor x_{i+1,j+1,a_5} \lor x_{i+1,j+2,a_6})
\]
\[ \phi_{\text{move}} = \bigwedge_{1 \leq i, j \leq n^k} \text{ (the (i, j) window is “legal”) } \]

the (i, j) window is “legal” =

\[ \equiv \bigwedge_{(a_1, \ldots, a_6) \text{ ISN’T “legal”}} \neg x_{i,j,a_1} \lor \neg x_{i,j+1,a_2} \lor \neg x_{i,j+2,a_3} \lor \neg x_{i,j+1,a_4} \lor \neg x_{i+1,j,a_5} \lor \neg x_{i+1,j+2,a_6} \]

\[ O(n^{2k}) \] clauses
Summary. Our goal was to prove:
Every A in NP has a polynomial time reduction to 3SAT

For every A in NP, we know A is decided by some nondeterministic \( n^k \) time Turing machine N.

We gave a generic method to reduce N and a string \( w \) to a 3CNF formula \( \phi \) of \( O(|w|^{2k}) \) clauses such that

*satisfying assignments to the variables of \( \phi \) directly correspond to accepting computation histories of N on w.*

The formula \( \phi \) is the AND of four 3CNF formulas:

\[
\phi = \phi_{cell} \land \phi_{start} \land \phi_{accept} \land \phi_{move}
\]
Theorem (Cook-Levin): 3SAT is NP-complete

Corollary: 3SAT ∈ P if and only if P = NP

Given your favorite problem \( \Pi \in \text{NP} \), how can we prove it is NP-hard?

Generic Recipe:
1. Take a problem \( \Sigma \) that you know to be NP-hard (e.g., 3SAT)
2. Prove that \( \Sigma \leq_p \Pi \)

Then for all \( A \in \text{NP} \), \( A \leq_p \Sigma \) and \( \Sigma \leq_p \Pi \)
We conclude that \( A \leq_p \Pi \), therefore \( \Pi \) is NP-hard
Π is NP-Complete
Reading Assignment

Read Luca Trevisan’s notes for an alternative proof of the Cook-Levin Theorem!

Sketch:
1. Define CIRCUIT-SAT: *Given a logical circuit* \( C(y) \), *is there an input* \( a \) *such that* \( C(a)=1 \)?

2. Show that CIRCUIT-SAT is NP-hard:
   *The* \( n^k \times n^k \) *tableau for* \( N \) *on* \( w \) *can be simulated using a logical circuit of* \( O(n^{2k}) \) *gates*

3. Reduce CIRCUIT-SAT to 3SAT in polytime

4. Conclude 3SAT is also NP-hard